データ並列言語における多重ループの統一的計算分散方式

太田 寛[†] 西谷 康仁[†]

分散メモリ向けデータ並列言語のコンパイラにおける重要機能の 1 つは，データの分散に基づいて計算処理をプロセッサに分散する計算分散機能である．計算分散については，従来から 1 次元配列と 1 重ループを対象として多くの研究が行われてきたが，多次元配列や多重ループに対する計算分散は，まだ十分に研究されているとはいえない．本研究では，配列添字とループ制御変数が 1 対 1 に対応しない場合などを含む一般的な多重ループに対して，統一的に計算分散を実施する方式を提案する．提案方式は，マッピング標準形による計算マッピングの表現方法，与えられた多重ループに対する計算マッピングの決定方法，計算マッピングに対するループ変換方法から構成される．本方式を様々な多重ループに適用し，これらが統一的に取り扱えることを実例で示す．本方式を実装し，様々な配列添字を含むループや NAS Parallel ベンチマークの FT を対象として評価を行った結果，いずれの場合にもスケーラブルな性能向上が得られることを確認した．

A Unified Method of Computation Distribution for Nested Loops in Data-parallel Languages

Hiroshi Ohta[†] and Yasunori Nishitani[†]

An important role of a compiler for distributed-memory data-parallel languages is the computation distribution, which distributes computation among processors based on the data distribution. Although computation distribution has been intensively studied for a 1-dimensional array and a single loop, it is not yet sufficiently studied for a multidimensional array and a loop nest. In this study, we present a unified method of computation distribution for a general loop nest, including the case where array subscripts and loop control variables do not correspond uniquely to each other. Our method consists of representing the computation mapping by *mapping normal form*, deciding computation mapping for a given loop nest, and transforming the loop nest based on the computation mapping. We apply the method to various loop nests and illustrate the unified treatment. We have implemented the method and carried out evaluation using loops with various array subscripts and FT of the NAS Parallel Benchmarks. In all the cases, scalable performance is achieved.

1. はじめに

分散メモリ型マルチプロセッサ向けのプログラミング言語として，HPF（High Performance Fortran）[7] などの様々なデータ並列言語が提案されている．データ並列言語のコンパイラは，データマッピング（データの各プロセッサへの割当て）に基づいて，計算マッピング（計算処理の各プロセッサへの割当て）を決定し，並列コードに変換する計算分散機能を備えている必要がある．

1 次元配列と 1 重ループを対象とする計算分散については，従来から多くの研究が行われてきている[1)~3)]．しかし，多次元配列や多重ループを対象とした計算分散は，まだ十分に研究されているとはいえない．多重ループには，単なる 1 重ループの組合せでは対処できない様々な場合が存在する．たとえば以下のような場合がある．

(1) 配列のある次元の添字がループ制御変数を含まない．
(2) 配列の各次元の添字がループ制御変数と 1 対 1 に対応しない．
(3) 配列が，あるプロセッサ次元に沿って，複数のプロセッサ上に複製されている（複製マッピング），または，単一のプロセッサ上にのみ存在する（シングルマッピング）．

これらのうち，(2) については，Kennedy ら[4]，Ramanujam ら[5]，Sato ら[9] の研究が知られている．しかし，これらの研究は特定の添字パターンを対象としており，様々な多重ループを統一的に扱うことはで

† 新情報処理開発機構マルチプロセッサコンピューティング日立研究室
RWCP Multiprocessor Computing Hitachi Laboratory

きなかった．また，Adve ら[6] は線形不等式系ソルバを利用した多重ループの変換方法を提案しているが，コンパイル時間が長い，プロセッサ数が可変のときに特殊な扱いが必要になるなどの問題点があった．

コンパイル時に計算分散できない場合は，実行時解決法（runtime resolution）という方式に基づくコードが生成されてしまう．これは，全プロセッサが元のループの繰返し範囲全体を実行し，配列参照のたびに毎回，参照される要素が自プロセッサに存在するかどうかを判定するというものであり，非常に実行性能が悪い．

本研究では，このような問題を解決するため，上記の (1)〜(3) をも含めた一般的な多重ループに対する計算分散方式を提案する．本方式では，配列の分散次元添字がループ制御変数の線形関数で表される場合には，ネストしたループのうち少なくとも 1 つは分散可能となり，実行時解決法を回避することができる．

以下，2 章では配列や計算のマッピングを表現するためのマッピング標準形を導入する．3 章では与えられた多重ループに対して計算マッピングを決定するアルゴリズムを述べ，様々な多重ループに対する適用の実例を示す．4 章では決定された計算マッピングに基づいてループを変換するアルゴリズムを述べる．5 章では変換されたループの実行性能を示す．6 章では関連研究との比較を述べる．

2. マッピング標準形

本研究では，配列マッピングのパターンとしては HPF 2.0 の核部分で規定されているものを対象とする[7]．HPF の公認拡張で規定されている GEN_BLOCK 分散や INDIRECT 分散は対象としない．なお，本方式の基本的な考え方自体は，HPF に限らず様々なデータ並列言語に適用可能なものである．

HPF での一般的な配列マッピングは，テンプレート（メモリ上に領域をとらない仮想的な配列）を介して 2 段階で行う．すなわち，まず配列をテンプレートに対して整列（align）させ，次にテンプレートを多次元のプロセッサ構成の上に分散（distribute）させる．この 2 段階マッピングはプログラム記述の際には便利な点もあるが，テンプレートに関して多少の冗長性を含み，そのままではコンパイラ内部でのマッピング表現方法としては適当でない．

そこで本方式では，配列マッピングの表現方法として，**マッピング標準形**を導入する．これは，配列マッピングを表す様々なパラメータの中から，主にテンプレートに関する冗長な情報をできるだけ除去して，ループ分散のために本質的なパラメータだけを抽出して整理したものである．マッピング標準形は**表 1** に示すパラメータから構成される．また，**表 2**，**表 3** に proc_axis_type および proc_axis_info パラメータの意味を示す．

表 1 マッピング標準形
Table 1 Mapping normal form.

(a) プロセッサ構成の形状	
proc_rank	プロセッサ構成の次元数
proc_size	プロセッサ構成の各次元の寸法(次元ごとに 1 つ)
(b) 配列の形状	
rank	配列の次元数
size	配列の各次元の寸法（次元ごとに 1 つ）
(c) プロセッサ次元ごとマッピング情報（各次元ごとに 1 セット）	
proc_axis_type	NORMAL, REPLICATED, SINGLE のいずれかの値．意味は表 2 を参照
proc_axis_info	proc_axis_type の値により，表 3 の意味を持つ
(d) 配列次元ごとマッピング情報*（各次元ごとに 1 セット）	
is_collapsed	当該次元が分散されていなければ TRUE，分散されていれば FALSE
axis_map	当該次元のマッピング先であるプロセッサ次元
align_lb	当該次元の最初の要素が整列するテンプレート要素のインデックス．ただしテンプレートの下限を 0 とする
align_stride	当該次元のテンプレートへの整列ストライド
blocksize	当該次元に対するテンプレート次元の分散ブロックサイズ

* is_collapsed が TRUE のときは (d) に分類される他のパラメータは使用しない．

表 2 proc_axis_type の内容
Table 2 Description of proc_axis_type.

値	意味
NORMAL	当該プロセッサ次元に沿って，配列のある次元が分散されている
REPLICATED	当該プロセッサ次元に沿って，配列が複製マッピングされている
SINGLE	当該プロセッサ次元上の単一のプロセッサに，配列がシングルマッピングされている

表 3 proc_axis_info の内容
Table 3 Description of proc_axis_info.

proc_axis_type の値	proc_axis_info の意味
NORMAL	対応する配列次元
REPLICATED	使用せず
SINGLE	配列を持つプロセッサのインデックス．ただし下限を 0 とする

なおマッピング標準形は，分散形式が block か cyclic かを表す明示的パラメータを含まない．これは blocksize などの他のパラメータから求められるからである．

本方式では，計算マッピングの表現方法としてもマッピング標準形を用いる．すなわち，多重ループ内のある文に着目したとき，その文のインスタンス空間を配

列のように見なすことによって，その文の計算マッピングをマッピング標準形で表すことができる．表 1 において，

配列 → 多重ループ全体
配列次元 → 文を囲む個々のループ

と読み替えればよい．表 2 の REPLICATED はそのプロセッサ次元に沿った全プロセッサによる重複実行を意味し，SINGLE はそのプロセッサ次元について単一プロセッサのみでの実行を意味することになる．

さらに計算マッピングに対するマッピング標準形では，以下の拡張を許すことにする．

(1) ある内側ループに対する size，align_lb，および align_stride は，その内側ループで不変である限り，多重ループ全体では可変であってもよい．
(2) proc_axis_type が SINGLE である場合の proc_axis_info（マッピング先プロセッサインデックス）は，文のインスタンスごとに可変であってもよい．

この拡張の効果は，次章以降で明らかになる．

3. 計算マッピングの決定方法

本章では，与えられた多重ループに対して，計算マッピングを決定する方法を示す．3.1 節で決定アルゴリズムを示し，3.2 節で各種ループの計算マッピングがどのように統一的に決定されるかを示す．

3.1 アルゴリズム

まず，計算マッピング決定の基準となる配列参照を 1 つ選択する．この基準配列参照のオーナプロセッサがその計算を実行するように，計算マッピングを決定する．以降の例では，いわゆる Owner-Computes Rule に基づいて左辺の配列参照を基準として選択しているが，他の方法，たとえば HPF の ON HOME 指示文で指定された配列参照を選択する方法であっても，本方式は同様に適用できる．

なお，基準配列参照以外の配列参照に対しては一般に通信が必要となるが，本稿では計算分散のみを主題とし通信は考えないことにする．通信については今後の別報告にて述べる予定である．

次に基準配列参照に基づいて計算マッピングを決定する．この手順の概要は次のとおりである．

(1) あるループの制御変数が配列分散次元添字に線形式の形で現れる場合には，その配列次元に対応するプロセッサ次元に沿ってループを分散する．
(2) その他のループは分散しない．
(3) ループと未対応のプロセッサ次元については，配列マッピングや添字に基づいて複製マッピングまたはシングルマッピングとする．

```
1:  for (基準配列参照を含む文のインスタンス空間の
      各次元 ds につき) do
2:    if (添字が F*Is+D の形であるような配列分散
        次元がある) then
        /* ここで,
           Is: 次元 ds に対するループの制御変数
           F: 非ゼロ定数
           D: そのループで不変な式
        */
3:      is_collapsed(ds) = FALSE;
4:      該当する分散次元から 1 個を選び，当該添字に
        従って次元 ds のマッピング情報を設定;
5:    else
        /* Is が配列分散次元添字に現れない,
           または現れるが F*Is+D の形でない */
6:      is_collapsed(ds) = TRUE;
7:    endif
8:  endfor
9:  for (プロセッサ構成の各次元 dp につき) do
10:   if (上の for ループで既にインスタンス空間次元と
      対応がついている) then
11:     proc_axis_type(dp) = NORMAL;
12:     proc_axis_info(dp) =
            対応するインスタンス空間次元;
13:   else if (配列のどの分散次元とも
        対応しない) then
        /* このとき配列の proc_axis_type(dp) は
           REPLICATED か SINGLE */
14:     proc_axis_type(dp) =
            配列マッピングの proc_axis_type (dp);
15:     proc_axis_info(dp) =
            配列マッピングの proc_axis_info(dp);
16:   else
        /* 配列のある分散次元 da と対応する．*/
17:     proc_axis_type(dp) = SINGLE;
18:     proc_axis_info(dp) = 配列次元 da の添字値の
            マッピング先プロセッサインデックス;
19:   endif
20: endfor
```

図 1 計算マッピング決定アルゴリズム
Fig. 1 Algorithm for deciding computation mapping.

詳細なアルゴリズムを**図 1** に示す．アルゴリズムの入力は多重ループおよび基準配列参照，出力は計算マッピング標準形である．

アルゴリズム中で，is_collapsed(*ds*) のように *ds* を添字とするパラメータは，インスタンス空間の第 *ds* 次元に対するものである．また，*dp* を添字とするパラメータは，プロセッサ構成の第 *dp* 次元に対するものである．

また，アルゴリズムの 4 行目のマッピング情報設定は，具体的には以下のようなものである．

axis_map(ds) = axis_map(da);
align_lb(ds) = align_stride(da) *
(F*L_loop + D - L_ary) + align_lb(da);
align_stride(ds) = align_stride(da) * F * S_loop;
blocksize(ds) = blocksize(da);

ここで，da は当該添字 (F*Is+D) を持つ配列次元であり，axis_map(da) などは配列マッピング標準形における第 da 次元のパラメータを表す．L_loop，S_loop はループの下限と増分，L_ary は配列の第 da 次元の宣言下限である．

3.2 各種ループへの適用

本節では，上記のアルゴリズムによって，様々なループの計算マッピングがどのように決定されるかを示す．

最も普通の場合として，すべてのループ制御変数が配列次元と 1 対 1 に対応し，かつ添字が線形である場合を考える．例を**図 2** (a) に示す．i1 は配列の 1 つの分散次元（第 1 次元）に一次式の形で現れるので，図 1 の 2 行目で then 節が適用され，i1 ループはプロセッサ構成の第 1 次元上に分散される．なおアルゴリズムの 4 行目では「該当する分散次元から 1 個を選び」となっているが，この場合はそのような次元が 1 つしかないので，選択の余地はないことに注意．同様に i3 ループは，プロセッサ構成の第 2 次元上に分散される．i2 ループは，i2 を添字に含む配列次元はあるが（第 2 次元），それが非分散なので図 1 の 5 行目以降の else 節が適用され，非分散 (is_collapsed(ds) = TRUE) となる．

アルゴリズムの後半の for において，プロセッサ構成の第 1，第 2 次元に対して 11，12 行目が適用され，それぞれ，i1 ループ，i3 ループに対する NORMAL となる．

結局，**図 3** の計算マッピング標準形が得られる．図 3 においてプロセッサ構成やインスタンス空間の各次元のパラメータは第 1 次元を先頭にして横に並べてある．'-' はそのパラメータを使用しないことを表す．

次に図 2 (b) のループを考える．この場合は，配列 a のマッピングに複製マッピングやシングルマッピングが含まれている．これにともない，計算マッピングにも，複製マッピングやシングルマッピングが含まれるようになる．

本ループに図 1 のアルゴリズムを適用すると，後半の for において，プロセッサ構成の第 2，第 3 次元に対して 13〜15 行目が適用される．その結果，これらのプロセッサ次元のマッピングは，配列 a のマッピングに従うことになる．最終的に，**図 4** の計算マッピング標準形が得られる．proc_axis_type に REPLICATED や SINGLE が現れている．

```
(a)
     real a(0:99, 0:99, 0:99)
!HPF$ processors p(0:3, 0:3)
!HPF$ distribute a(block,*,block) onto p
     do i1 = 0,98
        do i2 = 0,99
           do i3 = 2,50
              a(i1+1,i2,2*i3-1) = ...
           enddo
        enddo
     enddo

(b)
     real a(0:99, 0:99)
!HPF$ processors p(0:3, 0:3, 0:3)
!HPF$ template t(0:99, 0:3, 0:3)
!HPF$ align a(k,*) with t(k,*,3)
!HPF$ distribute t(block,block,block) onto p
     do i1 = 0,98
        do i2 = 0,99
           a(i1,i2) = ...
        enddo
     enddo

(c)
     real a(0:99, 0:99, 0:99)
!HPF$ processors p(0:3, 0:3)
!HPF$ distribute a(block,*,block) onto p
     do i1 = 0,98
        do i2 = 0,99
           a(i1,i2,99) = ...
        enddo
     enddo

(d)
     real a(0:99, 0:99, 0:99)
!HPF$ processors p(0:3, 0:3)
!HPF$ distribute a(block,*,block) onto p
     do i1 = 0,98
        do i2 = 0,99
           a(i1, i2, i1+1) = ...
        enddo
     enddo

(e)
     real a(0:99, 0:99)
!HPF$ processors p(0:3, 0:3)
!HPF$ distribute a(block,block) onto p
     do i1 = 1,49
        do i2 = 0, 49
           a(i1-1, i2+i1) = ...
        enddo
     enddo
```

図 2 様々なループ
Fig. 2 Various loops.

図 2 (c) のように配列参照のある分散次元の添字が定数となっている場合は，シングルマッピングとして扱われる．ここで，i1，i2 ループは，(a) の場合と同様にアルゴリズム前半の for ループによって，それぞれ

proc_rank	2		
proc_size	4	4	
proc_axis_type	NORMAL	NORMAL	
proc_axis_info	1	3	
rank	3		
size	99	100	49
is_collapsed	FALSE	TRUE	FALSE
axis_map	1	-	2
align_lb	1	-	3
align_stride	1	-	2
blocksize	25	-	25

図 3 ループ (a) に対するマッピング標準形
Fig. 3 Mapping normal form for the loop (a).

proc_rank	3		
proc_size	4	4	4
proc_axis_type	NORMAL	REPLICATED	SINGLE
proc_axis_info	1	-	3
rank	2		
size	99	100	
is_collapsed	FALSE	TRUE	
axis_map	1	-	
align_lb	0	-	
align_stride	1	-	
blocksize	25	-	

図 4 ループ (b) に対するマッピング標準形
Fig. 4 Mapping normal form for the loop (b).

proc_rank	2	
proc_size	4	4
proc_axis_type	NORMAL	SINGLE
proc_axis_info	1	3
rank	2	
size	99	100
is_collapsed	FALSE	TRUE
axis_map	1	-
align_lb	0	-
align_stride	1	-
blocksize	25	-

図 5 ループ (c) に対するマッピング標準形
Fig. 5 Mapping normal form for the loop (c).

分散，非分散となる．プロセッサ構成の第 2 次元はインスタンス空間次元と対応せず，かつ配列の分散次元（第 3 次元）と対応しているので，アルゴリズムの 16 行目の else 節が適用されシングルマッピングとなる．

計算マッピング標準形は**図 5** のようになる．第 2 次元の proc_axis_type が SINGLE となっている．第 2 次元の proc_axis_info はマッピング先プロセッサインデックスが 99/25=3 であることを示している．ここで 25 は配列マッピングの blocksize である．

図 2 (d) は，あるループの制御変数が複数の分散次元に現れる場合である．この場合，アルゴリズムの 3, 4 行目によって，i1 ループは配列の第 1 次元の添字 ‘i1’ または第 3 次元の添字 ‘i1+1’ のいずれかに従って

proc_rank	2	
proc_size	4	4
proc_axis_type	NORMAL	NORMAL
proc_axis_info	1	2
rank	2	
size	49	50
is_collapsed	FALSE	FALSE
axis_map	1	2
align_lb	0	i1
align_stride	1	1
blocksize	25	25

図 6 ループ (e) に対するマッピング標準形
Fig. 6 Mapping normal form for the loop (e).

分散される．どちらを選ぶかはアルゴリズムでは規定されていないが，ここでは仮に第 1 次元が選ばれたとする．選ばれなかった配列第 3 次元のマッピング先であるプロセッサ第 2 次元は，アルゴリズム後半の for ループにおいて，インスタンス空間次元と対応しなくなる．そのため，アルゴリズムの 17 行目によってシングルマッピングとなる．proc_axis_info は配列インデックス ‘i1+1’ のマッピング先プロセッサインデックスであるから，(i1+1)/25 となる．

計算マッピング標準形は，第 2 次元の proc_axis_info が (i1+1)/25 となる点を除いて，ループ (c) のものと同一である．このようなマッピングが表現可能となることが，2 章の最後で述べた，SINGLE のときに proc_axis_info が文のインスタンスごとに可変であるのを許すという拡張の効果である．

図 2 (e) では，配列の第 2 次元の添字に複数のループの制御変数が混在している．アルゴリズム前半の for ループで，まず i1 ループについて，第 1 次元添字 ‘i1-1’ によって 3, 4 行目が適用される．このとき，第 2 次元添字 ‘i2+i1’ は，内側ループ制御変数 i2 が i1 ループ不変ではないので，F*Is+D の形とは見なされないことに注意．次に，i2 ループについて考えるときは，制御変数 i1 は i2 ループ不変式なので，第 2 次元添字 ‘i2+i1’ は F*Is+D の形と見なされ，やはりアルゴリズムの 3, 4 行目が適用される．したがって，両方のループが分散される．

計算マッピング標準形は**図 6** のようになる．第 2 次元の align_lb が i1 となっているが，これは i2=0 のときに配列の第 2 次元添字が i1 となることによるものである．これによって，i2 ループのマッピングが表現可能となっている．これは，2 章の最後で述べた，内側ループに対する align_lb が，多重ループ全体では可変である場合を許すことの効果である．

なお，これからも分かるように，配列の分散次元添字がループ制御変数の線形結合である場合，最低でも，

それらのループの内で最内側のものは分散できることになる．なぜなら，最内側ループにおいては，外側ループ制御変数はすべて不変なので，添字は F*Is+D の形と見なされるからである．

4. 計算マッピングに基づくループ変換

本章では，計算マッピングが与えられたときの，ループの変換方法を述べる．これは元のループを変換して，各プロセッサが実行する SPMD プログラムを生成するものである．手順の概要は次のとおりである．

(1) 分散ループの範囲を縮小してローカルな範囲とする．非分散ループの範囲は元のままとする．
(2) プロセッサ次元がシングルマッピングとなっている場合は，特定プロセッサのみが計算を実行するように，if 文で保護を掛ける．

なお (1) でローカル範囲を求める方法は，1 重ループに対する各種の従来方法[1),2)] が適用できる．ループ (e) のように align_lb などのパラメータに変数が含まれている場合でも，内側ループについては不変であるから定数の場合と同様に扱える．

詳細なアルゴリズムを**図 7** に示す．以下，前章の図 2 (d) の例を用いてアルゴリズムを説明する．

まずアルゴリズムの前半の for ループを考える．インスタンス空間の第 1 次元の is_collapsed は FALSE であるから，アルゴリズムの 5 行目により i1 ループの範囲が縮小される．一方，第 2 次元の is_collapsed は TRUE であるから，アルゴリズムの 3 行目により i2 ループの範囲は元のままとなる．

次に後半の for ループを考える．プロセッサ構成の第 1 次元の proc_axis_type は NORMAL であるから，この次元については何もしない．一方，第 2 次元の proc_axis_type は SINGLE であるから，第 2 次元のプロセッサインデックスが (i1+1)/25 のプロセッサのみが実行するように，if 文を生成してループボディに保護を掛ける．

変換後のプログラムを**図 8** (a) に示す．

ここで，L1，U1，S1 は縮小されたループ範囲である．また，if 文内の my_pindex(2) は自プロセッサの第 2 次元のインデックスを表す．なお，配列第 3 次元の添字は mod 演算によりローカルインデックス化されている．

シングルマッピングに対する簡単な最適化として**保護移動**がある．これは，マッピング先プロセッサが多重ループ全体で不変ならば，if 文による保護を多重ループの外へ移動するものである．たとえば，前章の図 2 (c) では proc_axis_info(2)=3 で不変であるから，

```
 1: for(インスタンス空間の各次元 ds につき) do
 2:   if (is_collapsed(ds) == TRUE) then
 3:     ループ範囲を元のままとする;
 4:   else
 5:     当該次元 ds のマッピングにしたがって
        ループ範囲を縮小する;
 6:   endif
 7: endfor
 8: for (プロセッサ構成の各次元 dp につき) do
 9:   if (proc_axis_type(dp) == SINGLE) then
10:     proc_axis_info(dp) で指定される
        プロセッサだけが実行するように
        ループボディに保護を掛ける;
11:   else /* NORMAL または REPLICATED */
12:     何もしない;
13:   endif
14: endfor
```

図 7 ループ変換アルゴリズム
Fig. 7 Algorithm for loop transformation.

```
(a)
   real a(0:24, 0:99, 0:24)
   do i1 = L1,U1,S1
      do i2 = 0,99
         if(my_pindex(2) == (i1+1)/25) then
            a(i1, i2, mod(i1+1,25)) = ...
         endif
      enddo
   enddo

(b)
   real a(0:24, 0:99, 0:24)
   if(my_pindex(2) == 3) then
      do i1 = L1,U1,S1
         do i2 = 0,99
            a(i1, i2, 24) = ...
         enddo
      enddo
   endif
```

図 8 変換後プログラム
Fig. 8 Transformed programs.

保護移動が適用できる．このとき，変換後のプログラムは図 8 (b) のようになる．

5. 性能評価

3.2 節では提案方法によって様々なループが分散可能になることを示したが，本章ではループが分散されることによる性能向上効果を示す．このために，提案方式のプロトタイプを我々の開発した HPF 処理系[8)] 上に実装し，日立 SR2201 上で実機評価を行った．評価用プログラムとしては，本評価のために作成したカーネルループと，NAS Parallel ベンチマーク（NPB）[13)] の FT の主要ループを用いた．

5.1 カーネルループ

まず，3 章で述べたような様々な配列添字を含む，

```
ループ 1          ループ 2          ループ 3
real a(N,N)       real a(N,N)       real a(N,N)
do i = 1,N        do j = 1,N/2      do i = 1,N
  a(i,i)= ...     do i = 1,N/2        a(1,i) = ...
enddo               a(i+j,j)= ...   enddo
                  enddo
                  enddo
```

図 9 評価用カーネルループ
Fig. 9 Kernel loops for evaluation.

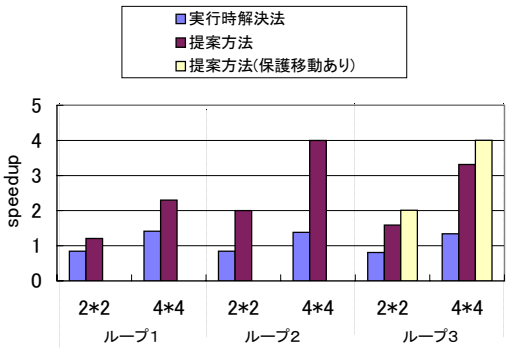

図 10 カーネルループの性能向上比
Fig. 10 Speedup for the kernel loops.

図 9 の 3 種類のカーネルループを用いて評価を行った．いずれの場合も配列は (block,block) で分散した．また，N の値は，ループ 1，ループ 3 では 5000，ループ 2 では 2000 とした．代入文の右辺の計算は，文の 1 インスタンスあたりの演算量が保護のオーバヘッドと同程度になるように調節した．これは実行時解決法でもある程度の性能向上が得られるような条件下での比較を行うためである．通信の影響を除くため，右辺の変数は全プロセッサ上に複製し，通信が発生しないようにした．

上記プログラムを，実行時解決法および提案方法の 2 通りでコンパイルし，生成されたプログラムの性能を測定した．またループ 3 については，提案方法において保護移動を行ったものも測定した．プロセッサ構成は 2*2 および 4*4 とした．

図 10 に元のプログラムを逐次実行した場合に対する性能向上比を示す．すべての場合において，提案方法は実行時解決法に比べて良い性能が得られている．その比率は最小で 1.4 倍（ループ 1，2*2），最大で 2.9 倍（ループ 2，4*4）である．また，ループ 3 では保護移動によりさらに 20〜25%の性能向上が得られている．これらのプログラムでは，ループの性質上，プロセッサ数が P*P のときには P 倍の性能向上が理論的限界であるが，ループ 2 やループ 3 では，ほぼその限界に到達する性能が得られている．

```
1:  complex*16 x(nx,ny,nz),xout(nx,ny,nz)
2:  complex*16 y(nblock, nz, 2)
3:  do k = 1, nz
4:     do jj = 0, ny - nblock, nblock
5:        do j = 1, nblock
6:           do i = 1, nx
7:              y(j,i,1) = x(i,j+jj,k)
8:           enddo
9:        enddo
10:       call cfftz (y)
11:       do j = 1, nblock
12:          do i = 1, nx
13:             xout(i,j+jj,k) = y(j,i,1)
14:          enddo
15:       enddo
16:    enddo
17: enddo
```

図 11 NPB/FT の主要ループ
Fig. 11 Kernel loop of NPB/FT.

なおループ 1 では，いずれか一方のプロセッサ次元についてはループ内に保護コードが残るため，ループ 2 やループ 3 に比べて性能向上が低い．このような場合については，文献 4)，5)，9) などに述べられている方法との併用によって，さらに性能を改善できると考えられる．

5.2 NPB/FT

次に，NPB2.3-serial の FT の主要ループを用いて評価を行った．このループはサブルーチン cffts1 内のもので，元の逐次ループの概要は**図 11** のとおりである．第 2 次元に関して計算をブロック化しているため，下線部に j+jj という添字が現れている．ループ内で最も計算負荷の大きい部分は，10 行目で呼び出されているサブルーチン cfftz である．

このループでは，配列 xout を (*,*,block) と 1 次元分散すれば，従来方法でも k ループが分散されて簡単に性能向上が得られる．しかし本評価では，添字 j+jj が分散次元に現れる場合の効果を見るために，あえて配列 xout を (*,block,block) で分散した．前と同様に，通信の影響を除くため右辺の配列 x は全プロセッサ上に複製とした．また，配列 y は k ループ，jj ループの new 変数とした．また，サブルーチン cfftz に対して，我々の HPF 処理系がサポートする local_pure 指示文[8] を指定することによって，10 行目の手続呼び出しがループ分散の妨げとならないようにした．local_pure 指示文は，手続きが副作用を持たず，かつその手続き内のデータに対して分散指示がないことを処理系に教える指示文である．処理系は，このような手続きを数学関数などの組み込み手続きと同様に扱うことができ，ループを分散しても安全であると判断できる．

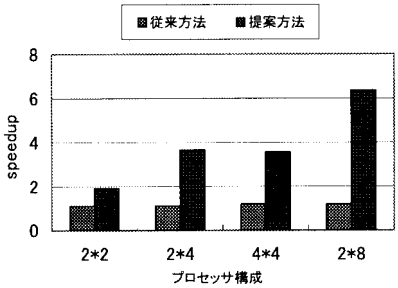

図 12 NPB/FT の性能向上比
Fig. 12 Speedup for NPB/FT.

この多重ループに対して提案方法を適用すると，3行目の k ループと 11 行目の j ループが分散される．一方，ベースにした HPF 処理系の従来方法では，添字 j+jj のためにどのループも分散されず，実行時解決法になってしまう．

図 12 に元のプログラムを逐次実行した場合に対する性能向上比を示す．計時区間はこの主要ループの部分だけである．配列サイズは 128*128*32 とし，プロセッサ構成は 2*2，2*4，4*4，2*8 の 4 通りを用いた．

従来方法は，全プロセッサにおいて元のループ範囲を実行し，特に cfftz の呼び出し回数が逐次実行と同じなので，ほとんど性能向上が見られない．一方，提案方法では，プロセッサ数の増加にともなう性能向上が得られている．ただし，cfftz の呼び出しは，プロセッサの第 2 次元に沿ってのみ分散され，第 1 次元に沿っては複製となるため，第 2 次元のプロセッサ数を増加にほぼ比例した性能向上となる．

以上をまとめると，カーネルループや FT において，提案方式により，スケーラブルな性能向上が得られることが確認された．

6. 関連研究

複雑な配列添字を含む多重ループの分散方法は，文献 4)，5)，9) で提案されている．これらの研究では，(i,i)，(i+j) などの配列添字が現れた場合に，各プロセッサで実行すべきイタレーション集合を求める方法を扱っている．しかしこれらの方法は，一般的な配列マッピングや配列添字が与えられた場合に対する統一的な方法ではなかった．これらの方法は，本提案方法が対象とする様々な多重ループの中の，特殊な場合に対する最適化と見なすことができる．

Adve ら[6] は線形不等式系ソルバを利用した多重ループの変換方法を提案している．これは，かなり広範な場合に対して適用可能であるが，プロセッサ数が可変の場合などには非線形式が生じ，特殊な扱いが必要だった．また，一般的な線形不等式ソルバを利用しているため，コンパイル時間が長いという問題点があった．

多重ループの計算マッピングについては，文献 10)，11) で扱われている．これらは人手指示に基づくものであり，コンパイラによる計算マッピング決定方法は述べられていない．また，複雑な配列添字の扱いについても述べられていない．

本研究で用いたマッピング標準形と類似のマッピング表現方法は，文献 12) で用いられている．本研究のマッピング標準形の特徴は，2 章の最後に述べた拡張によって，様々な計算マッピングが統一的に表現できることである．

7. おわりに

分散メモリ向けデータ並列言語のコンパイラにおける，一般的な多重ループに対する計算分散方式を提案した．提案方式は，マッピング標準形による計算マッピングの表現方法，与えられた多重ループに対する計算マッピングの決定方法，計算マッピングに対するループ変換方法から構成されることが特徴である．本方式により，配列添字がループ制御変数と 1 対 1 に対応しない場合などを含む様々な場合が統一的に取り扱えるようになる．

本方式のプロトタイプを我々の開発した HPF コンパイラ上に実装し，様々な配列添字を含むループを用いて評価を行った．本方式により生成されたプログラムの実行性能は，従来の実行時解決法に比べて，最小で 1.4 倍，最大で 2.9 倍向上した．また，簡単な最適化によって，さらに 20〜25%の性能向上が得られた．また，NAS Parallel ベンチマークの FT の主要ループにおいても，提案方法により，プロセッサ数の増加にともなう性能向上を得られた．

本稿では計算分散について扱ったが，データ並列コンパイラのもう 1 つの重要機能は通信生成である．これについても今後報告していく予定である．また今回は単純なカーネルループと FT のみについて評価を行ったが，今後，計算分散と通信生成を組み合わせた方式を実アプリケーションやさらに多くのベンチマークなどに適用し，有効性を検証していく．また計算分散に関しては，5.1 節のループ 1 のようにループ内に保護コードが残るケースについて，これまでに提案されている各種最適化方法との併用により，さらなる改善を検討していく．

謝辞 本研究の機会を与えてくださった新情報日立研究室の菊池純男氏，布広永示氏，また本方式の実装に関してご助言をいただいた（株）日立製作所ソフト

ウェア事業部，同システム開発研究所，および日立東北ソフトウェア（株）の諸氏に感謝いたします．

参 考 文 献

1) Gupta, S.K.S., Kaushik, S.D., Huang, C.-H. and Sadayappan, P.: Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines, *J. of Parallel and Distributed Computing*, Vol.32, pp.155–172 (1996).
2) van Reeuwijk, K., Denissen, W., Sips, H.J. and Paalvast, E.M.R.M.: An Implementation Framework for HPF Distributed Arrays on Message-Passing Parallel Computer Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.7, No.9, pp.897–914 (1996).
3) Ramanujam, J., Dutta, S., Venkatachar, A. and Thirumalai, A.: Advanced Compilation Techniques for HPF, *7th International Workshop on Compilers for Parallel Computers* (1998).
4) Kennedy, K., Nedeljkovic, N. and Sethi, A.: Efficient Address Generation for Block-Cylic Distributions, *Int'l Conf. on Supercomputing '95*, pp.180–184 (1995).
5) Ramanujam, J., Dutta, S. and Venkatachar, A.: Code Generation for Complex Subscripts in Data-Parallel Programs, *10th Workshop on Languages and Compilers for Parallel Computing* (1997).
6) Adve, V. and Mellor-Crummey, J.: Using Integer Sets for Data-Parallel Program Analysis and Optimization, *SIGPLAN '98 Conf. on Programming Language Design and Implementation*, pp.186–198 (1998).
7) High Performance Fortran Forum,: *High Performance Fortran Language Specification Version 2.0*, Rice University (1997).
8) 佐藤真琴，太田 寛，布広永示：HPF トランスレータ Parallel FORTRAN の開発と評価，情報処理，Vol.38, No.2, pp.105–108 (1997).
9) Sato, M., Hirooka, T., Wada, K. and Yamamoto, F.: Program Partitioning Optimizations in an HPF Prototype Compiler, *COMPSAC'96*, pp.124–131 (1996).
10) 岩下英俊，進藤達也，岡田 信：VPP Fortran：分散メモリ型並列計算機向けプログラミング言語，情報処理学会論文誌，Vol.36, No.7, pp.1542–1550 (1995).
11) 末広謙二，草野和寛，蒲池恒彦，妹尾義樹，田村正典，左近彰一，渡辺幸光，白戸幸正：HPF 処理系における計算処理マッピング，*JSPP'95*, pp.369–376 (1995).
12) Offner, C.D.: A Data Structure for Managing Parallel Operations, *Proc. 27th Annual Hawaii Int'l Conf. on System Sciences*, pp.33–44 (1994).
13) Bailey, D., Harris, T., Saphir, W., van der Wijngaart, R., Woo, A. and Yarrow, M.: The NAS Parallel Benchmarks 2.0, Technical Report NAS-95-020, NASA Ames Research Center (1995).
http://science.nas.nasa.gov/Software/NPB/

(平成 11 年 8 月 31 日受付)
(平成 11 年 12 月 2 日採録)

太田 寛（正会員）

1962 年生．1987 年東京大学大学院理学系研究科地球物理学専門課程修了．同年，(株）日立製作所入社．現在，同社システム開発研究所主任研究員．入社以来，論理型言語の研究を経て，並列化コンパイラの研究に従事．1997 年より新情報処理開発機構マルチプロセッサコンピューティング日立研究室兼務．並列処理ソフトウェア全般，並列アーキテクチャに興味を持つ．電子情報通信学会，ACM，IEEE 各会員．

西谷 康仁（正会員）

1971 年生．1994 年東京大学工学部電子工学科卒業．同年，(株）日立製作所入社．現在，同社ソフトウェア事業部勤務．並列化コンパイラの研究開発に従事．1997 年より新情報処理開発機構マルチプロセッサコンピューティング日立研究室兼務．